

METHOD AND SYSTEM FOR STARTING OPERATING SYSTEM ENVIRONMENT

Patent number: JP6324849

Publication date: 1994-11-25

Inventor: RAWSON III FREEMAN L; SOTOMAYOR GUY G JR

Applicant: INTERNATL BUSINESS MACH CORP <IBM>

Classification:

- International: G06F9/06; G06F9/445

- european:

Application number: JP19940107472 19940425

Priority number(s):

Also published as:

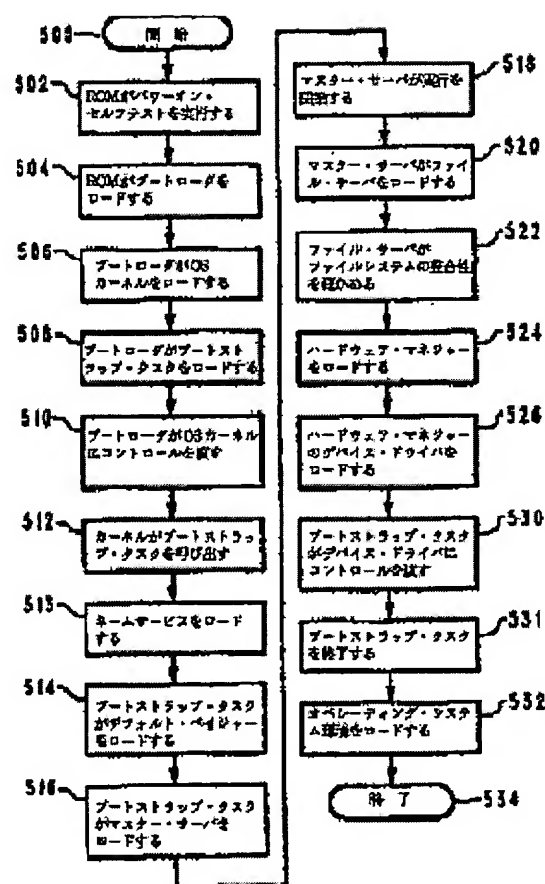
EP0622731 (A2)

EP0622731 (A3)

Abstract of JP6324849

PURPOSE: To provide an operating system environment.

CONSTITUTION: In a data processing system having an OS kernel and plural device drivers separated from the OS kernel, steps for (1) preparing a boot volume having the plural files of data structure constituted from plural files on a storage device and parameter for identifying the positions in a memory and (2) for loading plural programs related to the initialization of the operating system environment by using the boot volume and initializing the operating system environment are provided.



Data supplied from the esp@cenet database - Worldwide

BEST AVAILABLE COPY

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平6-324849

(43) 公開日 平成6年(1994)11月25日

(51) Int.Cl. ¹	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 9/06	4 1 0 D	9367-5B		
9/445		9367-5B	G 0 6 F 9/ 06	4 2 0 G

審査請求 有 請求項の数12 F D (全 15 頁)

(21) 出願番号 特願平6-107472

(22) 出願日 平成6年(1994)4月25日

(31) 優先権主張番号 08/054, 117

(32) 優先日 1993年4月26日

(33) 優先権主張国 米国 (US)

(71) 出願人 390009531
 インターナショナル・ビジネス・マシーンズ・コーポレーション
 INTERNATIONAL BUSINESS MACHINES CORPORATION
 アメリカ合衆国10504、ニューヨーク州アーモンク (番地なし)

(72) 発明者 フリーマン エル、 ローソン、 サード
 アメリカ合衆国 33487-2242 フロリダ州 ボカラトン ウッドビューテラス 17762

(74) 代理人 弁理士 合田 潔 (外3名)

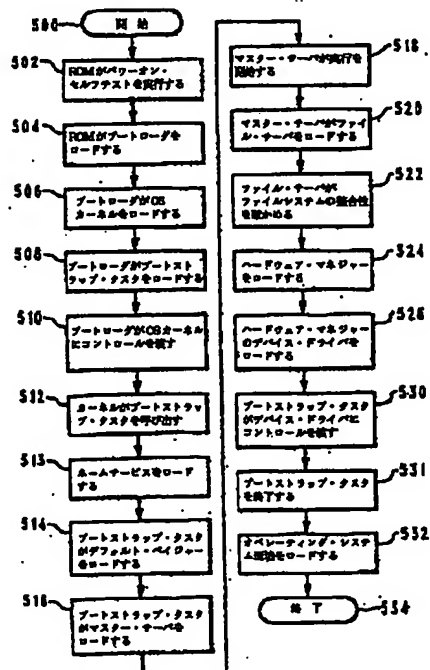
最終頁に続く

(54) 【発明の名称】 オペレーティング・システム環境の起動方法およびシステム

(57) 【要約】

【目的】 オペレーティング・システム環境を起動する方法を提供する。

【構成】 OSカーネルと、OSカーネルから分離されている複数のデバイス・ドライバを持つデータ処理システム上で、(1)記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意し、(2)上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードして、オペレーティング・システム環境の初期化を行う、ステップを有するオペレーティング・システム環境起動方法。



【特許請求の範囲】

【請求項1】 カーネルと上記カーネルから分離されている複数のデバイス・ドライバ・プログラムを備えたデータ処理システムでオペレーティング・システム環境を起動させる方法であって、

記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意し、
上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードして、オペレーティング・システム環境の初期化を行う、
ステップを有する方法。

【請求項2】 上記パラメータが、
上記ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を有し、ファイルを構成するデータ構造が少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされる請求項1に記載の方法。

【請求項3】 カーネルと上記カーネルから分離されている複数のデバイス・ドライバ・プログラムを備えたデータ処理システムでオペレーティング・システム環境を起動させる方法であって、ブート・ボリュームをメモリに用意するステップを有し、上記ブート・ボリュームが上記ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を備え、ファイルを構成するデータ構造が少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされ、さらに上記方法が、
上記ブート・ボリュームを使用してオペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードする、
ステップを有し、オペレーティング・システム環境の初期化が行われる方法。

【請求項4】 上記のロードするステップが、デバイス・ドライバをロードすることを含む請求項3に記載の方法。

【請求項5】 マスター・サーバをロードすることをさらに含む請求項4に記載の方法。

【請求項6】 ファイル・サーバおよびファイル・シス

テムをロードすることをさらに含む請求項5に記載の方法。

【請求項7】 上記ファイル・システムがログと複数のファイルを有し、上記ファイルを使う前に、上記ファイルに訂正が必要であるか否かを判断するためにログを使用することをさらに含む請求項6に記載の方法。

【請求項8】 デバイス・ドライバをロードした後に、オペレーティング・システム環境をロードすることをさらに含む請求項7に記載の方法。

【請求項9】 オペレーティング・システム環境と、OSカーネルと、複数のデバイス・ドライバを持ち、上記複数のデバイス・ドライバがカーネルと分離されているデータ処理システムであって、
デバイス・ドライバ機能を除くカーネルと、
記憶装置に記憶されているブート・ボリュームと、
ブートストラップ・タスクと、
を有し、上記ブート・ボリュームが、
ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、
上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、
記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、
を有し、ファイルを構成するデータ構造が、少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされ、さらに上記ブートストラップ・タスクが上記ブート・ボリュームを使用してデータ記憶装置からファイルをメモリにロードすることによりオペレーティング・システム環境が初期化されるシステム。

【請求項10】 ブート・ボリュームにより保証されて、データ処理システムが正しくない可能性のあるファイルでなく、正確なファイルを使用することができる請求項9に記載のシステム。

【請求項11】 オペレーティング・システム環境と、OSカーネルと、複数のデバイス・ドライバを持ち、上記複数のデバイス・ドライバが上記カーネルと分離されているデータ処理システムであって、
記憶装置上の複数のファイルから構成されるデータ構造の複数のファイルおよびその位置を識別するパラメータを有するブート・ボリュームをメモリに用意する手段と、
上記ブート・ボリュームを使用して、オペレーティング・システム環境の初期化に関連する複数のプログラムを上記メモリにロードする手段と、
を有することによりオペレーティング・システム環境の初期化を行うシステム。

【請求項12】 上記パラメータが、
ブート・ボリュームに関連しているファイルを探す最初のポイントを示すファイル・アンカーと、

上記ブート・ボリュームに関連しているファイルの識別レコードを持ち上記ファイル・アンカーに関連している複数のファイルヘッダと、記憶装置中の連続しているブロックを識別する複数のブロック・リスト・エレメントと、を有し、ファイルを構成するデータ構造が、少なくとも1つの上記ブロック・リスト・エレメントを使用して識別されアクセスされる、請求項11に記載のシステム。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、マイクロ・カーネルを使用してデータ処理システムを初期化する方法とシステムに関する。

【0002】

【従来の技術】オペレーティング・システムは、データ処理システムのオープンシステム標準において、優劣を競う分野になっている。国内・海外を問わず業界におけるこの分野の競争の結果、1つの統合的なオペレーティング・システムが複数のオペレーティング・システムをサポートすることが望ましいとされるようになった。

【0003】オペレーティング・システムを作成する1つの手法に、基本的なハードウェア資源を制御するオペレーティング・システムの部分と、オペレーティング・システム環境のユニークな特徴を定義する部分とを分離して、「純粋なカーネル」を作る方法がある。ハードウェア資源に関連する部分はオペレーティング・システム・カーネル（以下OSカーネルまたはカーネルと略す）と呼ばれることが多い。このような設計を採用すると、1つのデータ処理システムの上で複数のオペレーティング・システムを動かすことが可能になる。また、同一のデータ処理システム上で、複数のオペレーティング・システムをネイティブモードで同時に動かすことが可能になる。

【0004】しかし、純粋なカーネルを作成するのに複雑な問題は、既にあるオペレーティング・システムに依存せず、かつ、純粋なカーネルが異種のオペレーティング・システム環境をサポートするようにすることである。例えば、オペレーティング・システムの純粋カーネルは、いくつかの異種のオペレーティング・システム環境に対して中立であり、そのカーネル（即ちマイクロ・カーネル）をブートストラップするためにユーザ・レベルでデバイス・ドライバを使用でき、かつ、その上に少なくとも1つのオペレーティング・システムを作動させなければならない。

【0005】現在、この問題の解決は、OSカーネルとそのオペレーティング・システムで実行されるプログラムとを結びつけるか、あるいは、システムが、ブートのすぐ後でロードされるオペレーティング・システム環境を持ち、ブートストラップをしている間に使用できるファイル・システムを持っていると仮定することで対処し

ている。しかし、デバイス・ドライバとブートストラップ中あるいはその後でロードされるタスクとが分離されている環境では、この解決策は動かない。

【0006】

【発明が解決しようとする課題】したがって、ハードウェアで定義したROMブートの終了から、最初のオペレーティング・システム環境の標準的な初期化の実行までの連続的な手順の全てに対処できるブートストラップのアーキテクチャが求められる。

【0007】

【課題を解決するための手段】本発明は、マイクロ・カーネルを使用してデータ処理システムを初期化する方法とシステムを提供することを目的とする。

【0008】本発明は、オペレーティング・システム環境と、OSカーネルと、カーネルから分離されている複数のデバイス・ドライバ・プログラムを持っているデータ処理システムに関するものである。このシステムでは、カーネルはデバイス・ドライバ機能を持たず、デバイス・ドライバはユーザ・レベルのタスク即ちプログラムである。

【0009】記憶装置にブート・ボリュームが用意され、ブート・ボリュームはファイル・アンカーと、複数のファイルヘッダと、複数のブロック・リスト・エレメントを持っている。ファイル・アンカーはブート・ボリュームに関連するファイルを探すための始まりのポイントを示し、ファイル・ヘッダーはアンカーに関連し、それぞれがブート・ボリュームに関連しているファイルの識別を持ち、ブロック・リスト・エレメントはそれぞれ記憶装置の中の1つの連続したデータのブロックを識別し、ファイルを構成するデータ構造が、少なくとも1つのブロック・リスト・エレメントを使用して識別されアクセスされる。ブートストラップ・タスクにより、データ記憶装置からメモリに、ブート・ボリュームを使用してファイルがロードされ、ファイルがアクセスできるようになり、オペレーティング・システム環境が初期化される。

【0010】

【実施例】図1に、本発明が実施できるデータ処理システムを示す。コンピュータ50が、システム・ユニット52、ディスプレイ装置54、キーボード56、およびマウス58を備えている。コンピュータ50は、IBM PS/2あるいはIBM RISC SYSTEM/6000等あるいは同等の、本発明の実施に適したシステムである。

【0011】図2に、本発明が実施できるコンピュータ50の主要な構成要素を示す。システムユニット52は各種の構成要素を相互接続するためのシステムバス60を持つことが望ましい。マイクロプロセッサ62はシステムバス60（例えばマイクロチャネル・システムバス）に接続され、また、数値計算用のコプロセッサ64を持つてもよい。また、ダイレクト・メモリ・バス（DMA）

コントローラ66がシステムバス60に接続され、これにより、大量の入出力データ転送中に、各種のデバイスがマイクロプロセッサ62からのサイクル割り当てが可能になる。

【0012】ROM68とRAM70もシステムバス60に接続されている。ROM68はパワー・オンの際のセルフテストを含んでいる。CMOS RAM72がシステムバス60につながり、システム構成情報を持つことも可能である。

【0013】また、システムバス60にはメモリ・コントローラ74、バス・コントローラ76、インタラプト・コントローラ78が接続されていて、各種の周辺装置、アダプタおよびデバイス間のデータの流れをシステムバス60を介して制御する。システムユニット52は、さらに各種の入出力コントローラ、即ち、キーボード/マウス・コントローラ80、ビデオ・コントローラ82、並列コントローラ84、直列コントローラ86、ディスク・コントローラ88を持っている。キーボード/マウス・コントローラ80は、キーボード90とマウス92のためのハードウェア・インターフェースである。ビデオ・コントローラ82は、ディスプレイ装置94のハードウェア・インターフェースである。並列コントローラ84は、印刷装置96のような装置のハードウェア・インターフェースである。直列コントローラ86は、モデム98のような装置のハードウェア・インターフェースである。ディスク・コントローラ88は、フロッピーディスク・ユニット100のハードウェア・インターフェースである。ハードディスク・ユニット104のインターフェースを提供するディスク・コントローラ102のような拡張カードをシステムバス60に加えてもよい。空きスロット106を使って、システム52に他の周辺装置やアダプタをつけ加えることもできる。

【0014】図2に示したハードウェアは、具体的な用途によって変わり得る。例えば、既に述べたハードウェアに加えまたはその代わりに、光ディスク媒体、音声アダプタ、PALやEPRMのようなプログラム用のチップ等の周辺デバイスを用いることもできる。

【0015】図3に、本発明を用いてデータ処理システムをブートする方法のフローチャートを示す。ブート・ボリューム、ブートストラップ・タスク、OSカーネル、マスター・サーバ等の要素は図4から図11で詳しく述べる。プロセスはブロック500で始まりブロック502に進む。ブロック502はパワー・オン・セルフテストを行うROMの機能である。ブロック504はブート・ローダをロードするROMの機能を示す。ブロック506はOSカーネルをロードするブート・ローダを示す。ブロック508はブートストラップ・タスクをロードするブート・ローダを示す。

【0016】ブロック510で、ブート・ローダはOS

カーネルにコントロールを渡す。ブロック512で、OSカーネルがブートストラップ・タスクを呼び出す。ブロック513でネーム・サービスがロードされる。ブロック514で、ブートストラップ・タスクがデフォルト・ペイジャーをロードする。次のブロック516で、ブートストラップ・タスクがマスター・サーバをロードする。プロセスは次に、マスター・サーバが実行を始めるブロック518に進む。ブロック520で、マスター・サーバがファイル・サーバをロードする。次に、ブロック522で、ファイル・サーバがファイルシステムの整合性を確かめる。次にプロセスは、ハードウェア資源マネージャをロードするブロック524に進む。さらに、ハードウェア資源マネージャのデバイス・ドライバをロードするブロック526に進む。

【0017】次にプロセスは、ブロック530で、ブート用デバイスのコントロールを、ブートストラップ・タスクから、ブロック526でロードされたデバイス・ドライバに渡す。ブートストラップ・タスクはブロック531で終了する。その後、プロセスは、OS環境をロードして初期化するブロック532に進む。プロセスはその後ブロック534で終了する。

【0018】ブート・プロセスでは、データ処理システムのオペレーティング・システムを起動するために、いくつかの異なるブート要素が使用される。これらのブート要素には、ブート・ボリューム、ブート・ローダ、OSカーネル、ブートストラップ・タスク、ネーム・サービス、デフォルト・ペイジャー、イニシャル・ページ・スペース、マスター・サーバ、ファイルシステム・サーバ、ハードウェア資源マネージャ、および、ブート・デバイスドライバのOS環境がある。ブート・ボリュームは、ブート・プロセスでどのファイルやプログラムをロードすべきかを示す記述を持っている。ブート・ボリュームはディスク上の一連の逐次的セクターからなっている。本発明では、ブート・ボリュームはロードすべきファイルの記述だけを持ち、ファイルそのものは持たない。ブート・ボリュームにファイルを持たないことにより、ファイル内容を持った場合に比べて、ブート・ボリュームを小さくできる。さらに、ブート・ボリュームの内容を、標準的なユーティリティー・プログラムをどれか1つのOS環境で実行して容易に変更できる。こうすることにより、ブート・ボリュームの内容を変更し、いまの内容よりも大きなスペースが要する場合に、ブート・ボリュームに十分なスペースを割り当てることができる。

【0019】今日、多くのファイル・システムはログを持っていて、システム故障の際のデータの損失を防ぐようにしている。故障したファイルシステムのデータはアクセスできないようにし、ユーティリティー・プログラムがログを適用しファイルシステムを初期化し、データ構造を直してからファイルシステムが使えるようにして

いる。ブート・ボリュームは、ファイルシステムと修正ユーティリティ・プログラムを実行するのに十分なもう1組のデータ構造を持ち、これにより、ファイルシステム動作回復を可能にしている。

【0020】あるシステムでは、ブート・ボリュームが、ブート・ボリュームに記述されている全てのファイルを持っている場合もあろう。どのように具体化するかは、ブートするときにハードウェアがどのように作動するかに依存している。例えば、RS/6000では、ハードウェアの構成が完了するまで入出力動作は行われない。

【0021】ブート・ボリュームで記述されているファイルをブート・ボリュームが持っていないシステムでは、本発明を使用すれば、通常の動作でこれらのファイルを読むユーティリティ・プログラムやOS環境プログラムが、故障して正しくない可能性のあるファイルでなく正しいファイルを読むことができる。

【0022】図4に、本発明によるブート・ボリュームの構造のダイアグラムを示す。ブート・ボリューム298には通常レコードと呼ばれるデータ構造、即ち、ブート・ボリューム・ファイル・アンカー300、ファイルヘッダ302、および、ブロック・リスト・エレメント306がある。実際のファイルはファイルシステムに記憶され、ブート・ボリュームの設計とレイアウトはファイルシステムに依存しない形で作られる。

【0023】図4に矢印で示しているが、通常ブート・ボリューム内容は全て同時に作成され、個々のデータ構造はリンクされたリストとしてよりも、アレイとして記憶される。ブート・ボリュームにある全てのオフセットは、ブート・ボリュームの始めからの相対的なバイトオフセットである。また、本発明により、全ての複数バイトの値は、データを解釈するシステムのエンディアンそのままのエンディアン (natural endian) として記憶される。

【0024】図5に、ブート・ボリューム・ファイル・アンカー300のダイアグラムを示す。ブート・ボリューム・ファイル・アンカー300は、ブート・ボリューム298の内容を総合したアンカーとしてのレコードである。アンカーは、ブート・ボリューム298で記述されたファイルを探すためのスタートポイントの役をする。アンカー300にはレコードのタイプを記述するフィールド310があり、フィールド310はこのレコードをブート・ボリューム・ファイル・アンカーとして識別する。フィールド312は、ブート・ボリューム・ファイル・アンカー300が占めるバイト数を表す。次のフィールド314は、ブート・ボリューム・ファイル・アンカーの現在のバージョン番号を示す。このフィールドが、ブート・ボリューム・ファイル・アンカー300のフィールドの構造や意味の変更を表す。

【0025】フィールド316は、ブート・ボリュームが占めるバイト数を表す。バイト数はブート・ボリュー

ム・ファイル・アンカーを含む。フィールド318はブート・ボリュームの現在のバージョン番号を示し、ブート・ボリュームのフィールドの構造や意味の変更を表す。フィールド320は、ブート・ボリューム・ファイル・アンカーに含まれているファイルヘッダの数を表す。フィールド322はそれぞれブート・ボリュームが参照している個々のファイルを記述している。フィールド322については後で詳しく述べる。

【0026】図6に、ファイルヘッダ322のダイアグラムを示す。ファイルヘッダは、ブート・ボリュームで使われるデータ構造で、データ処理システムに実際のファイルシステムがロードされる前に、ブート・プロセス中に参照されるそれぞれのファイルを記述している。ファイルヘッダ322にフィールド330があり、このレコードがファイルヘッダ322であると識別するフォーマットを示す。フィールド332は、ファイルヘッダが占めるバイト数を示すフィールドである。フィールド332の数は、最初のフィールドからブロック・リスト・エレメントの最後までを含む。

【0027】フィールド334は、ファイルヘッダのバージョン、即ち、ファイルヘッダ構造の現在のバージョン番号を示し、ファイルヘッダ322のフィールドの構造や意味の変更を表す。フィールド336は、ファイルが占める全バイト数、即ちファイルの大きさを示す。フィールド338はファイルの名前を示し、ナル (null) で終わるストリングである。典型的には、フィールド338には、実際のユーザのファイルシステムの中のファイルの名前が入る。フィールド338の大きさは通常256キャラクタである。フィールド340は、ファイルヘッダ322で記述されたファイルのタイプを表す。以下のファイルタイプがフィールド340に記述される。

【0028】(1)OSカーネル。ファイルがOSカーネルであることを示すフラッグ。これにより、ブート・ローダ・プログラムはこの属性を持ったファイルを探し、OSカーネルの名前に永久に符号化しないようにする。

【0029】(2)ブートストラップ・タスク。ファイルがブートストラップ・タスクであることを示すフラッグ。これにより、ブート・ローダ・プログラムはこの属性を持ったファイルを探し、ブートストラップ・タスクの名前に永久に符号化しないようにする。

【0030】(3)ネームサービス。ファイルがネームサーバであることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ネームサービス・プログラムの名前に永久に符号化しないようにする。

【0031】(4)ペイジャー。ファイルがデフォルト・ペイジャー (即ち、まだ誰のものでもないメモリへのペイジャー) であることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ペイジャー・プログラムの名前に永久に符号化し

ないようにする。

【0032】(5)ペイジング・ファイル。ファイルがペイジング用のデフォルト・ペイジャーによって使われることを示すフラッグ。これにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、ペイジング・ファイルの名前に永久に符号化しないようにする。

【0033】(6)マスター・サーバ。ファイルがマスター・サーバであることを示すフラッグ。マスター・サーバは他のサーバをロードするプログラムである。このフラッグにより、ブートストラップ・タスクはこの属性を持ったファイルを探し、マスター・サーバの名前に永久に符号化しないようにする。

【0034】(7)マスター・サーバ構成ファイル。このファイルが、ブート・プロセスの1部として立ち上げるために、マスター・サーバが、システムと他のサーバを構成するのに必要な構成情報を持っていることを示すフラッグ。これにより、ブートストラップ・タスクはファイルを探し、そのファイルを、マスター・サーバが実行を始めるときにマスター・サーバに渡す。

【0035】(8)実行可能ファイル。ファイルが一般的な実行可能ファイルであることを示すフラッグ。マスター・サーバはこのフラッグを使って、ファイルがロードされるべきサーバかどうかを判断する。

【0036】(9)データファイル。ファイルが実行可能でないデータを持っていることを示すフラッグ。

【0037】フィールド342は次のファイルヘッダへのオフセットであり、ブート・ボリュームの中にある次のファイルヘッダのブート・ボリュームの始めからのオフセットを示す。このファイルヘッダがブート・ボリュームの中で最後のものである場合は、このフィールドは、本発明の具体化では、ゼロを持つ。フィールド344は、ファイルを表すのに必要なブロック・リスト・エレメントの数を示す。フィールド346は、ブロック・リスト・エレメントを示し、ディスク上で物理的に連続しているブロックの数を記述する。これについては、後で詳しく述べる。

【0038】図7に、ブロック・リスト・エレメント346を示す。ブロック・リスト・エレメント346は、ハード・ディスク・ドライブの上の、物理的に連続したブロックの一群を記述するデータ構造である。ブロック・リスト・エレメント346にはフィールド348があり、レコードのタイプ即ちレコードのフォーマットを示す。フィールド348は、このレコードがブロック・リスト・エレメント346の中の1つのブロック・リスト・エレメントであると識別する。フィールド350は、ブロック・リスト・エレメント346が占めているバイト数を表す。バイト数は最初のフィールドから数えられる。フィールド352は、ブロック・リスト・エレメントの最新のバージョン番号を示し、ブロック・リスト・エレメント346のフィールドの構造と意味の変更を表

す。フィールド354は、ディスクドライブ上の1つのブロックが持っているバイト数、即ちブロックの大きさを表す。

【0039】デバイス識別符号(以下ID)はフィールド356で示される。このフィールドは、デバイスが載っているデバイスを示し、ナルで終わる一連のストリングで表される。フィールド358は、ブロック・リスト・エレメント346が表すデバイス上での物理的に連続しているブロックの数を示す。フィールド360はオフセット・ポインタの大きさを示す。このフィールドは、デバイスフィールドの始めからのオフセットに含まれているビット数を表す。本発明の具体化では、オフセットの記憶に使われるバイト数はいつも4の倍数であるが、オフセットフィールドの全てのビットが意味を持つわけではない。フィールド362はデバイスの始めからのオフセットを示す。このフィールドは、このレコードで記述されている最初のブロックのデバイスの始めからのバイトオフセットを持っている。このフィールドの大きさはフィールド358で定義される。

【0040】ブート・ローダは、例えばROMのようなハードウェアあるいはシステムで定めた仕組みによってロードされるプログラムである。ブート・ローダはブート・ボリュームのフォーマットを解釈し、システム・メモリにOSカーネルとブートストラップ・タスクをロードする任務を持つ。また、ブート・ローダはOSカーネルの実行をスタートさせる。OSカーネルとブートストラップ・タスクをメモリにロードするのに加えて、ブート・ローダは、コントロールをOSカーネルに渡す前に、OSカーネルに以下に述べる情報を渡す。

【0041】図8に、OSカーネルに情報を渡す方法のフローチャートを示す。プロセスはブロック400で始まり、ブロック402に進み、OSカーネルにメモリマップ情報を渡す。メモリマップ情報は、OSカーネルに対して、メモリの物理的なレイアウトを記述するものである。メモリ全部が物理的に連続していないならば、メモリマップは異なるいくつかのメモリ領域で表される。メモリマップは、OSカーネルにとって必要な、領域の属性情報を持つ。

【0042】プロセスはブロック404に進み、メモリの中のブートストラップ・タスクの位置(物理的なアドレス)と大きさがOSカーネルに渡される。OSカーネルに渡される情報は、実行が始まるブートストラップ・タスクへのオフセットを含むこともある。さらに、ブートストラップ・タスクがいくつかの領域(即ちテキストとデータ)に分割されている場合には、それぞれの領域を記述する情報もOSカーネルの送られる。次にプロセスはブロック406に進み、ブート・デバイスのデバイスIDがOSカーネルに渡される。ブート・デバイスのデバイスIDは、ブート・ローダがOSカーネルとブートストラップ・タスクをロードしたブート・ボリューム

が入っているデバイスを指す。この情報がOSカーネルに提供されるのは、さらにこの情報をブートストラップ・タスクに渡すためである。

【0043】次に、プロセスはブロック408に進み、OSカーネルに、ブート・ボリュームのブート・デバイスのオフセットが渡される。オフセットは、ブート・デバイス上のブート・ボリュームの位置を教えるものである。この情報により、ブート・デバイスの正確なレイアウトがわからなくても、ブート・デバイス上の複数のブート・ボリュームおよびブート・プロセスの他の構成要素が実行できるようになる。この情報も、OSカーネルに渡され、さらにブートストラップ・タスクに渡される。プロセスはブロック410で終了する。

【0044】情報は、公知の方法により、メモリの中のいろいろなプログラムの間で渡される。例えば、パラメ

値のタイプ	シンタックス	例
INTTYPE	int	VAR = -1 または VAR = 1
UINTTYPE	uint	VAR = 1
CHARTYPE	char	VAR = c
STRINGTYPE	"..."	VAR = "abcdefg"
MULTIRANGETYPE	int1..int2 int1,int2	VAR = 1..4 VAR = 1,3 VAR = 1..4,8..10,12,17

OSカーネルに渡される実際のパラメータを以下に示す。

【0048】MEMORYMAP = x1..y1, x2..y2, x3..y3
これはシステムのメモリマップを記述している。複数の領域があり得るのは、多くのシステムの場合、メモリがある物理的なアドレスには「空き」があるからである。システムに存在するメモリを十分に記述するのに必要な領域が記述される。値は整数で与えられる。計算の基になる数は十進法である。値を"0x"で始めれば、値はヘキサデシマルとみなされる。

【0049】BOOTTASK_NUM_REGION = n

これはOSカーネルに与えられた領域の数を表しブートストラップ・タスクを記述するものである。値は整数である。

【0050】BOOTTASK_REGION = x..y, vs, "..."

これはブートストラップ・タスクのアドレス・スペースの個々の領域を表し、値は以下の意味を持つ。

【0051】x..yはメモリの物理的領域を記述する。値は整数である。計算の基は十進法である。値を"0x"で始めれば、ヘキサデシマルとみなされる。

【0052】vsは領域のバーチャル・スタート・アドレスである。ブートストラップ・タスクのアドレス・スペースの中の、領域の始まりを示すアドレスである。値は整数で十進法である。"0x"で始めれば、ヘキサデシマルとみなされる。

【0053】"..."は領域に適用されるべきプロテクション・フラッグで、一連のストリングで表される。スト

ータをスタックすること、即ち、メモリの中の情報を指すポインタを、その情報が行くべきプログラムに渡すことによって行える。

【0045】本発明の具体化により、情報は、OSカーネルに、ニューライン・キャラクター（即ち\n）で分離された連続したキャラクターのアレイとして提供できる。アレイ全体はナルキャラクターで終わる。アレイのフォーマットは以下になる。

【0046】variable1 = value 1 \ n

variable2 = value 2 \ n

...

variablen = value n \ n \ 0

変数(variables)の値には以下のタイプがある。

【0047】

例
VAR = -1 または VAR = 1
VAR = 1
VAR = c
VAR = "abcdefg"
VAR = 1..4
VAR = 1,3
VAR = 1..4,8..10,12,17

リングの内容はフラッグのシンボリック・ネームで、"I"の字で分離されている。シンボリック・ネームは以下のものがある。

VM_PROT_READ 読み取りアクセスができる。
VM_PROT_WRITE 書き込みアクセスができる。
VM_PROT_EXECUTE 実行アクセスができる。

【0054】BOOTTASK_STATE = n1, n2, ...nm

これはOSカーネルがブートストラップ・タスクの実行を始めたときにセットされる実行状態を示す。これによって表されるパラメータは、正しい実行状態を確立するための、使用するプロセッサに依存した情報である。

【0055】KERNEL_PATCH = x..y, value

これにより、OSカーネルの初期化中に、あるカーネルデータを変更することができる。例えば、デバッグのあるオプションを選択してオンにできる。値は以下に説明する。

【0056】x..yは変更したいカーネルのバーチャル・アドレスの範囲を表す。範囲はバイトで表すので、範囲としては複数のバイト値で指定しなければならない。範囲が何を表しているかは、値（以下に示す）のタイプによって解釈される。

【0057】valueはカーネルの複数のアドレスの範囲の中に置かれる値を表す。値は1つの値に限らず、全ての値が同じタイプであればリストにしてよい。

【0058】BOOT_DEVICE = "..."

これは、ブート・ボリュームが載っているデバイスを表し、デバイスの名前を含んだ一連のストリングである。

【0059】BOOTVOL_OFFSET = n

これは、ブート・デバイスの始まりからのブート・ボリュームの位置を表す。値は整数で十進法である。"0x"で始めれば、ヘキサデシマルとみなされる。

【0060】本発明では、ブートストラップ・タスクは以下の3つの主たる機能を持つ。即ち、(1)マスター・サーバも含めて、最初のいくつかのタスクをシステムにロードすること、(2)正式のシステム・レベルの機能がロードされるまで、暫定的なデバイス・ドライバ・セットの役をすること、(3)ブート・ボリュームのファイル・システム・サーバとして機能し、ブート・ボリュームにアクセスする必要のある他のタスクがブート・ボリューム自体の内容を解釈しなくてもすむようにすること。

【0061】情報はOSカーネルからブートストラップ・タスクに渡される。これは2つの方法により行われる。第1の方法は、情報をパラメータ（ブートストラップ・タスクが始まる時にスタックされている）として渡すことである。第2の方法は、情報をOSカーネルのブート・パラメータ・リストに載せ、ブートストラップ・タスクが取り出せるようにすることである。

【0062】図9のフローチャートに、OSカーネルからブートストラップ・タスクに情報を渡す方法を示す。プロセスはブロック420で始まり、ブロック422に進み、ブート・デバイスのデバイスIDがパラメータとしてブートストラップ・タスクに渡される。デバイスIDは、ロードされたOSカーネルとブートストラップ・タスクの入っているデバイスを識別するものである。

【0063】プロセスはブロック424に進み、ブート・デバイス上のブート・ボリュームがブートストラップ・タスクに渡される。ブート・ボリュームのオフセットは、ブート・デバイス上のブート・ボリュームの位置を示す。これにより、ブートストラップ・タスクはブート・ボリュームのフォーマット情報だけを持てばよく、ブート・デバイスの内容全体の情報を持たなくてすむ。プロセスはブロック426で終了する。

【0064】図10に示すように、ブートストラップ・タスクは、他に3つのプログラムをロードする。プロセスはブロック430で始まり、ブロック432に進み、システム・ネーム・サービスがロードされる。ブロック434で、デフォルト・ペイジャーがメモリにロードされる。ブロック436でマスター・サーバがメモリにロードされる。プロセスはブロック438で終了する。

【0065】システム・ネーム・サービス・プログラムは、ブートストラップ・タスクによってメモリにロードされる最初のプログラムである。ネーム・サービスが最初にロードされるのは、本発明により、ネーム・サービスと他のサーバとの接続をするためであり、ネーム・サービスをできるだけ早く起動させることにより、他のサーバがネーム・サービスを使えるようになる。これにより、暫定的なネーム・サービスが必要なくなる。

【0066】通常、デフォルト・ペイジャーはブート手順の早い段階でロードされ、メモリが十分とれるようにする。デフォルト・ペイジャーがあることにより、ブートストラップ・タスクによりデフォルト・ペイジャーに渡された最初のペイジング・スペースに記述されたブート・デバイス上の領域にペイジングが行えるようになる。その結果、比較的少ないメインメモリを持ったデータ処理システムでオペレーティング・システムが立ち上げられる。次にマスター・サーバがメモリにロードされ、これにより、ブートストラップ・タスクのローディング機能が不要になる。マスター・サーバは、それがブートストラップ・タスクによりロードされた後は、システム全体の動作を見る役をする。

【0067】マスター・サーバが起動すると、ブートストラップ・タスクは基本的なデバイス・ドライバの役割に変わり、ブート・デバイスの入出力の役割を担って、ペイジングやプログラムのローディングを行う。さらに、この時点で、ブートストラップ・タスクはマスター・サーバに対して、ユーザとの連絡の手段になるコンソールの役をする。適切なシステム機能が起動されると、ブートストラップ・タスクは持っている資源を全て放して終了する。ブートストラップ・タスクは1組のメッセージを持っていて、これにより、マスター・サーバが、ブートストラップ・タスクのある操作を実行するよう要求できるようになっている。

【0068】以上の結果、本発明により、ブート・ボリュームを使用した単純なファイル・システムが提供でき、ブートストラップ・タスクが、ブート・ボリュームに含まれているファイルにアクセスできるようになる。さらに、ブートストラップ・タスクがプログラム・ロードを提供するので、最初に必要ないくつかのプログラムがロードできる。プログラムをローディングする過程で、ブートストラップ・タスクにより、プログラムが、起動されたネーム・サービスにアクセスできるようになる。さらに、ブートストラップ・タスクが1組の簡単なデバイス・ドライバを提供するので、他のサーバが、ブート・ボリュームで記述されたブート・デバイス上にデータを読み/書きできるようになる。また、ブートストラップ・タスクは、ブート・プロセス中にサーバとオペレータないしユーザが連絡できるようにするための簡単なデバイス・ドライバを提供している。またさらに、ブートストラップ・タスクは、他のプログラムがブート・ボリュームの内容にアクセスできるように1組のファイルシステム・インターフェースを提供している。この結果、データ処理システムの実際のファイルシステムがロードされ起動されるまで、ブートストラップ・タスクはブート・ボリュームのファイルシステムの役をする。

【0069】図11にデータ処理システムをブートするのに使われる主要な要素をブロックダイアグラムにして示す。メモリ600は、OSカーネル602、ブートス

トラップ・タスク604、マスター・サーバ606、および、ハードディスク・デバイス・ドライバ608を持っている。OSカーネル602とブートストラップ・タスク604は、ブート・プロセス中にブート・ロードによりロードされる。ブートストラップ・タスクはブート・プロセスの一環としてマスター・サーバ606をロードする。最初、ブート・ボリュームからの情報を使って、ファイルがハードディスク・ドライブ610からブートストラップ・タスクによりロードされる。

【0070】ハードディスク・ドライブからのファイル10を要求するハウスキーピング機能と他のタスクは、OSカーネル602にメッセージを送ることにより、アクセスすることができる。この段階、即ち、十分なシステム機能がメモリ600にロードされるまで、OSカーネルはメッセージをブートストラップ・タスク604にまわす。主要な機能の内、特にハードディスク・デバイス・ドライバ608をメモリ600にローディングすることにより、マスター・サーバ606は、ブートストラップ・タスク604とブート・ボリュームを必要とすることなく、ハードディスク・ドライブ610にアクセスできる。20その時点で、ブートストラップ・タスク604は終了し、ハードディスク・ドライブ610のファイルにアクセスする要求は、OSカーネル602によりハードディスク・デバイス・ドライバ608にまわされる。ブートストラップ・タスク604が扱っていた他の機能も、メモリ600にある他のプログラムあるいはデバイス・ドライバ（図示せず）にまわされる。

【0071】上記の具体化はハードディスクを使用した30が、本発明は、データ処理システムをブート即ち初期化するのに使用できる他の媒体を使って具体化することもできる。

【0072】上記の具体化は、1台のコンピュータ上での具体化として説明したが、本発明はネットワーク、分散処理、あるいは、マルチプロセッサ・データ処理システムでも実施することができる。本発明の具体化は、ユーザ・レベルのデバイス・ドライバを持ち、Machの派生物のマイクロ・カーネルを使用するデータ処理システムで実施できるものである。Mach派生物のマイクロ・カーネルについての詳細は、「Mach: オペレーティング・システムの1つの基礎」(Richard Rashid et al., Mach: A Foundation For Operating Systems, Proceedings of the Second Workshop on Work Station Operating Systems, pp. 109-113, IEEE Computer Society, September, 1989)に記載されている。

【0073】

【発明の効果】上述したブート・ストラップのアーキテクチャにより、本発明は、ROMブートの終了から始まり、オペレーティング・システム環境の初期化の実行までの連続的な手段を提供することができる。

【図面の簡単な説明】

【図1】本発明が実施できるデータ処理システムを示す。

【図2】図1に示したデータ処理システムの、本発明を具体化するための主たる構成要素を示す。

【図3】本発明による、データ処理システムをブートする方法を示すフローチャート。

【図4】本発明のブート・ボリュームの構造を示すダイアグラム。

【図5】本発明のブート・ボリューム・ファイル・アンカーを示すダイアグラム。

【図6】本発明のファイルヘッダを示すダイアグラム。

【図7】本発明によるブロック・リスト・エレメントを示すダイアグラム。

【図8】本発明によるOSカーネルに情報を送る方法のフローチャート。

【図9】本発明による、OSカーネルからブート・ストラップ・タスクに情報を渡す方法のフローチャート。

【図10】本発明の、ブート・ストラップ・タスクによりプログラムをロードする方法のフローチャート。

【図11】本発明により、データ処理システムをブートするのに使用する主要構成要素を示すブロック図。

【符号の説明】

50	コンピュータ
52	システムユニット
54、94	ディスプレイ装置
56、90	キーボード
58、92	マウス
60	システムバス
62	マイクロプロセッサ
64	数値計算コプロセッサ
66	DMAコントローラ
68	ROM
70	RAM
72	CMOS RAM
74	メモリ・コントローラ
76	バス・コントローラ
78	インタラプト・コントローラ
80	キーボード/マウス・コントロ
ーラ	
82	ビデオ・コントローラ
84	並列コントローラ
86	直列コントローラ
88	ディスク・コントローラ
96	印刷装置
98	モデム
100	フロッピー・ディスクユニット
102	ディスク・コントローラ
104	ハードディスク・ユニット
298	ブート・ボリューム
300	ブート・ボリューム・ファイル

17

18

・アンカー

346

ブロック・リスト・エレメント

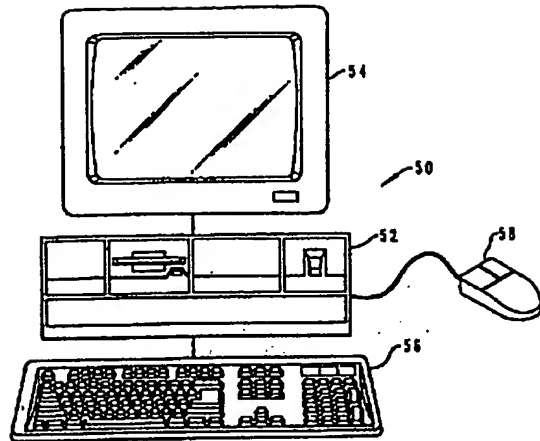
322

ファイルヘッダ

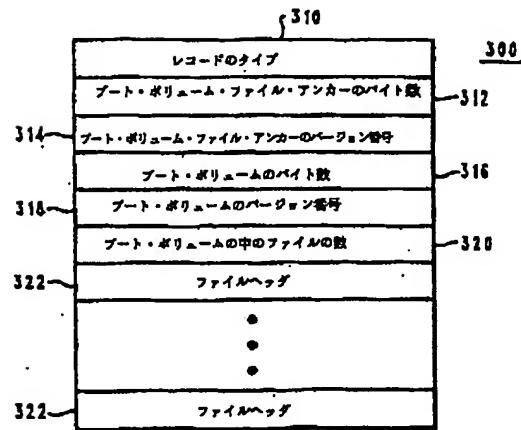
600

メモリ

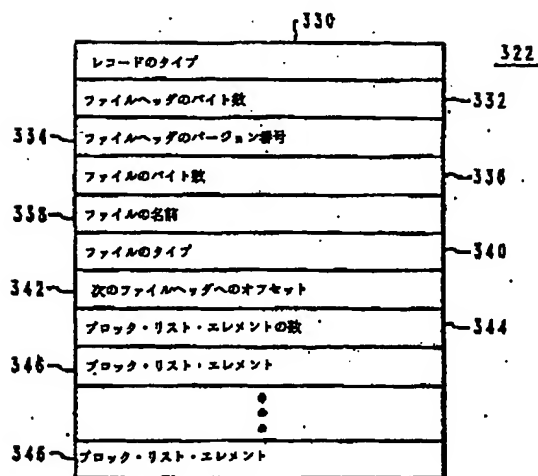
【図1】



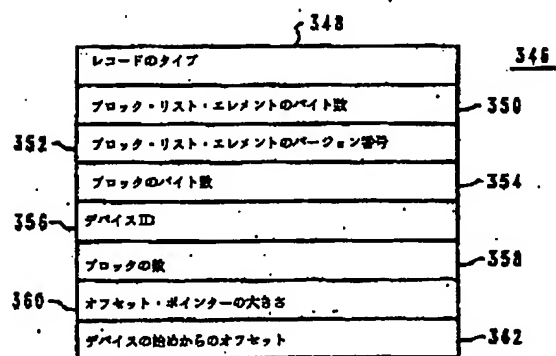
【図5】



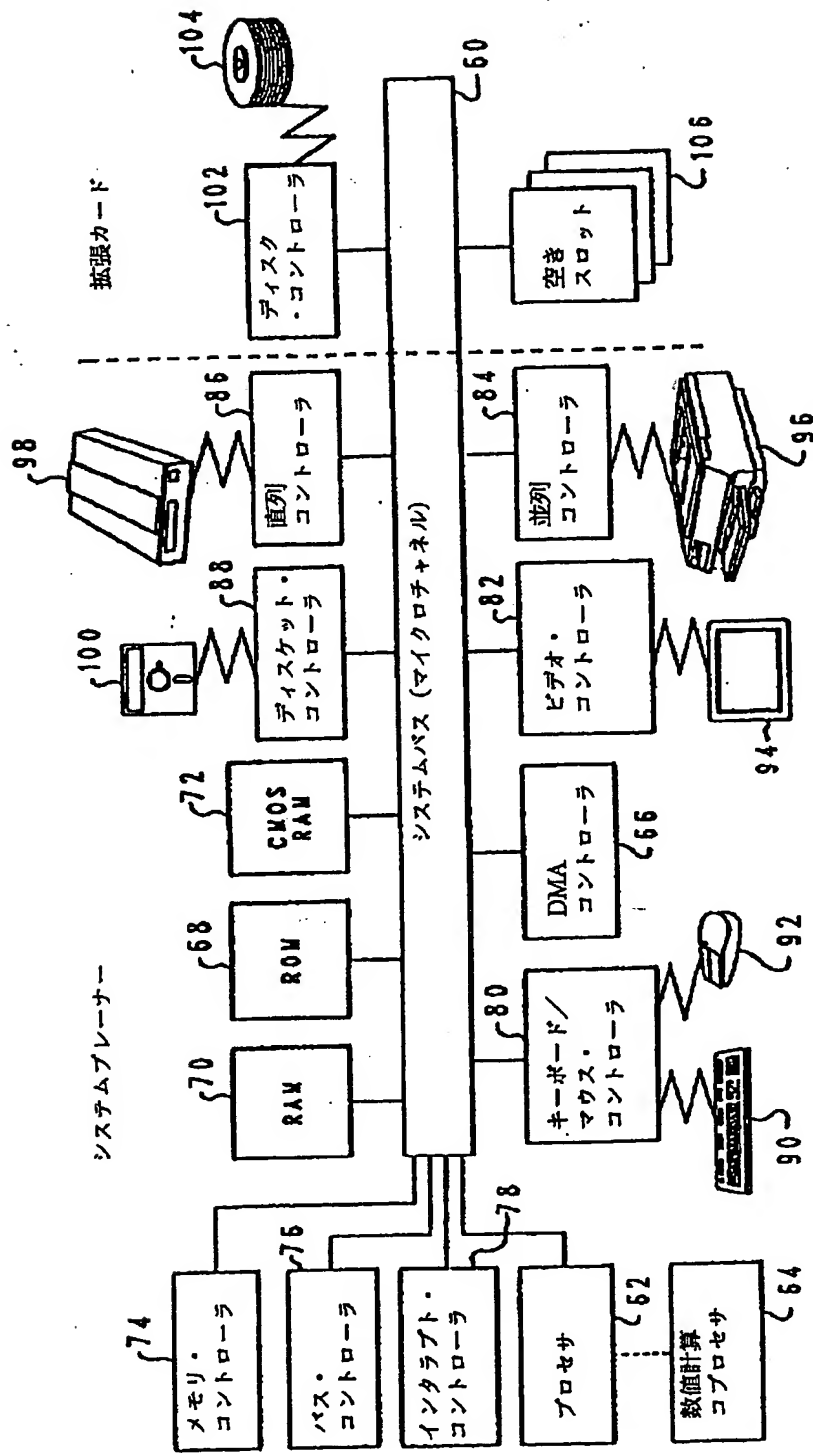
【図6】



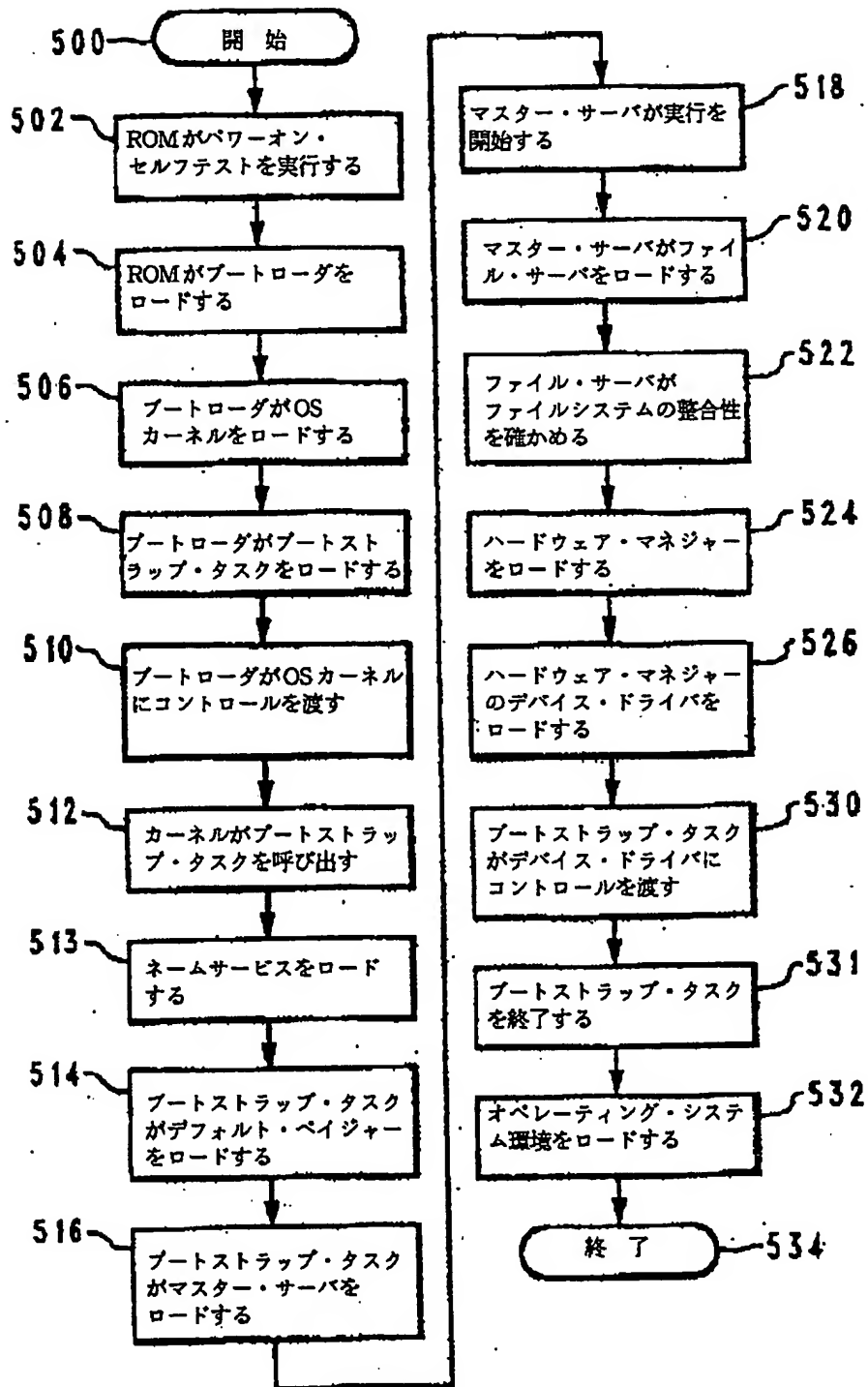
【図7】



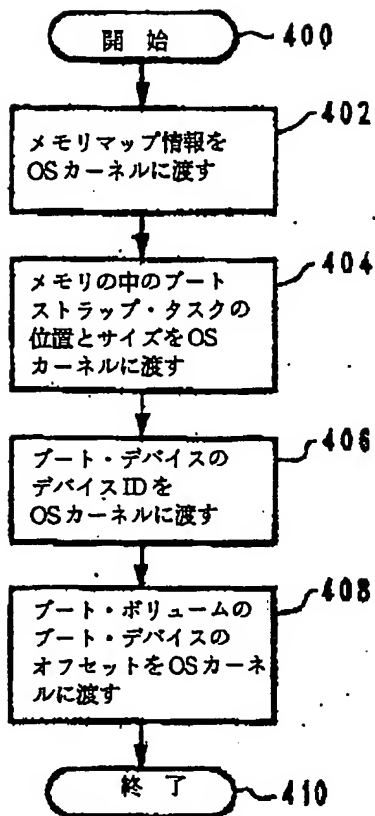
【図2】



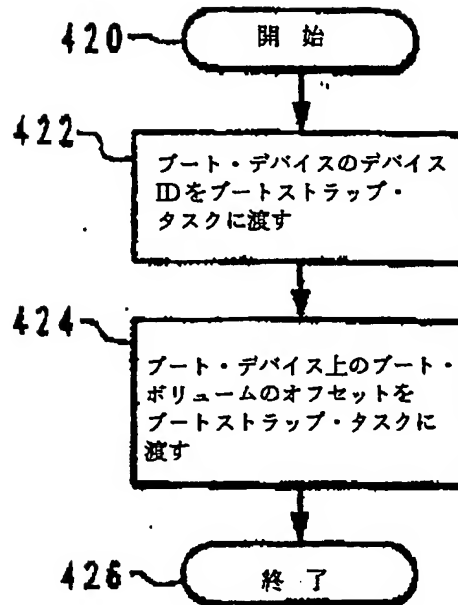
【図3】



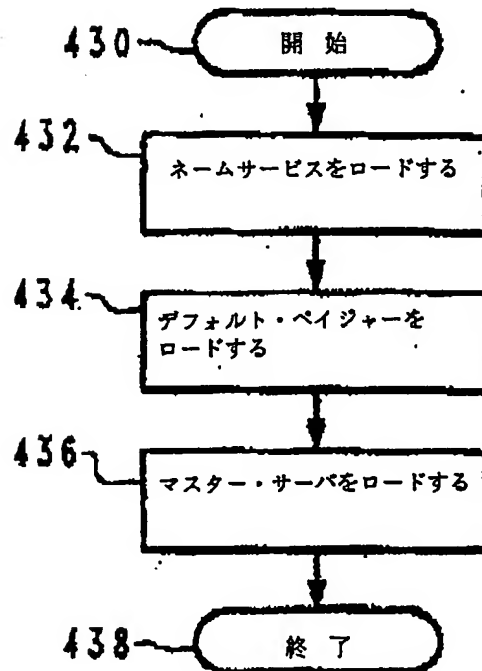
【図8】



【図9】



【図10】



**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☒ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.